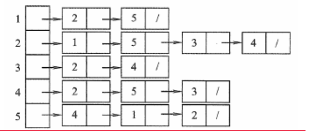
6.1图的基本概念

图的定义

- 图G由顶点集V和边集E组成,记为G=(V,E),其中 V(G)表示图G中顶点的有限非空集;E(G)表示图G中顶点之间的关系(边)集合
- $V=\{v_1, v_2, \cdots, v_n\}$ 则用|V|表示图G中顶点的个数也称图G的阶　$E=\{(u, v) \mid u \in V, v \in V\}$ 用|E|表示图G中边的条数
- 注意:线性表可以是空表,树可以是空树,但图不可以是空图
- 图的一些基本概念及术语
  - 有向图
    - 若E是有向边(也称弧)的有限集合时,则图G为有向图。
    - 弧是顶点的有序对,记为<v,w>, 其中v,w是顶点,v称为弧尾,w称为弧头,<v,w>称为从顶点v到顶点w的弧,也称v邻接到w, 或 w邻接自v
  - 无向图
    - 若E是无向边(简称边)的有限集合时,则图G为无向图
    - 边是顶点的无序对,记为(v,,w)或(w,v),因为(v,w)=(w,v),其中v,w是顶点
    - 可以说顶点w和顶点v互为邻接点。边(v,w)依附于顶点w和v,或者说边(v,w)和顶点y,w相关联
  - 简单图
    - 不存在重复边
    - 不存在顶点到自身的边，则称图G为简单图
  - 多重图
    - 若图G中某两个结点之间的边数多于一条，又允许顶点通过同一条边和自己关联，则G为多重图
  - 完全图(也称简单完全图)
    - 对于无向图,|E|的取值范围是0到n(n-1)/2,有n(n-1)2条边的无向图称为完全图,在完全图中任意两个顶点之间都存在边
    - 对于有向图,|E|的取值范围是0到n(n-1),有n(n-1)条弧的有向图称为有向完全图,在有向完全图中任意两个顶点之间都存在方向相反的两条弧
  - 子图
    - 设有两个图G=(V,E)和G'=(V',E'),若V'是V的子集,且E'是E的子集,则称G'是G的子图。若有满足V(G')=V(G)的子图G',则称其为G的生成子图
  - 连通、连通图和连通分量
    - 在无向图中,若从顶点v到顶点w有路径存在,则称v和w是连通的
    - 若图G中任意两个顶点都是连通的,则称图G为连通图,否则称为非连通图
    - 无向图中的极大连通子图称为连通分量
    - 若一个图有n个顶点,并且边数小于n-1,则此图必是非连通图
  - 强连通图、强连通分量
    - 强连通图：在有向图中,若从顶点v到顶点w和从顶点w到顶点v之间都有路径,则称这两个顶点是强连通的若图中任何一对顶点都是强连通的,则称此图为强连通
    - 有向图中的极大强连通子图称为有向图的强连通分量
  - 生成树、生成森林
    - 连通图的生成树是包含图中全部顶点的一个极小连通子图
    - 若图中顶点数为n,则它的生成树含有n-1条边
    - 对生成树而言,若砍去它的一条边,则会变成非连通图,若加上一条边则会形成一个回路
    - 在非连通图中,连通分量的生成树构成了非连通图的生成森林
  - 顶点的度，入度和出度
    - 度：定义为以该顶点为一个端点的边的数目
    - 无向图：顶点v的度是指依附于该顶点的边的条数　无向图的全部顶点的度的和等于边数的2倍
    - 有向图：全部顶点的入度之和与出度之和相等,并且等于边数
  - 边的权和网
    - 每条边都可以标上具有某种含义的数值,该数值称为该边的权值。这种边上带有权值的图称为带权图,也称网
  - 稠密图、稀疏图
    - 边数很少的图称为稀疏图,反之称为稠密图
    - 一般当图G满足 $|E|<|V|\log|V|$ 可以将G视为稀疏图
  - 路径、路径长度和回路
    - 路径：两个顶点之间相连的边
    - 路径长度:路径上边的数目称
    - 回路或环：第一个顶点和最后一个顶点相同的路径
    - 若一个图有n个顶点,并且有大于n-1条边,则此图一定有环
  - 简单路径、简单回路
    - 简单路径：顶点不重复出现的路径
    - 简单回路：除第一个顶点和最后一个顶点外,其余顶点不重复出现的回路
  - 距离
    - 从顶点u出发到顶点v的最短路径若存在,则此路径的长度称为从u到v的距离
    - 若从u到v根本不存在路径,则记该距离为无穷
  - 有向树
    - 一个顶点的入度为0、其余顶点的入度均为1的有向图,称为有向树

6.2图的存储及基本操作

- **邻接矩阵法**
  - 邻接矩阵存储
    - 是指用一个一维数组存储图中顶点的信息,用一个二维数组存储图中边的信息（即各顶点之间的邻接关系）
    - 存储顶点之间邻接关系的二维数组称为邻接矩阵
    - 结构示意
      - $A[i][j]=\begin{cases}1, & 若(v_i,v_j)或\langle v_i,v_j\rangle 是E(G)中的边\\ 0, & 若(v_i,v_j)或\langle v_i,v_j\rangle 不是E(G)中的边\end{cases}$
      - 带权图示意
        - $A[i][j]=\begin{cases}w_{ij}, & 若(v_i,v_j)或\langle v_i,v_j\rangle 是E(G)中的边\\ 0或\infty, & 若(v_i,v_j)或\langle v_i,v_j\rangle 不是E(G)中的边\end{cases}$
    - 邻接矩阵表示法的空间复杂度为O(n²),其中n为图的顶点数|V|
  - 特点
    - 无向图的邻接矩阵一定是一个对称矩阵（并且唯一）。因此,在实际存储邻接矩阵时只需存储上（或下）三角矩阵的元素
    - 对于无向图,邻接矩阵的第i行（或第i列）非零元素(或非0元素）的个数正好是第i 个顶点的度 $TD(v_i)$
    - 对于有向图,邻接矩阵的第i行（或第i列）非零元素（或非o元素）的个数正好是第i 个顶点的出度 $OD(v_i)$
    - 用邻接矩阵法存储图,很容易确定图中任意两个顶点之间是否有边相连。但是,要确定图中有多少条边,则必须按行、按列对每个元素进行检测,所花费的时间代价很大
    - 稠密图适合使用邻接矩阵的存储表示
- **邻接表法**
  - 当一个图为稀疏图时，使用邻接矩阵法要浪费大量的存储空间，而图的邻接表法结合了顺序存储和链式存储方法，减少了不必要的浪费
  - 结构
    - 对图G中的每个顶点建立一个单链表,第i个单链表中的结点表示依附于顶点 vi的边,这个单链表就称为顶点vi的边表（对于有向图则称为出边表）
    - 边表的头指针和顶点的数据信息采用顺序存储（称为顶点表）,所以在邻接表中存在两种结点:顶点表结点和边表结点
  - 示意图



  - 特点
    - 若G为无向图,则所需的存储空间为O(|V| + 2|E|);若G为有向图,则所需的存储空间为O|V|+|E|）
    - 对于稀疏图,采用邻接表表示将极大地节省存储空间
    - 在邻接表中,给定一顶点,能很容易地找出它的所有邻边,因为只需要读取它的邻接表
    - 在有向图的邻接表表示中,求一个给定顶点的出度只需计算其邻接表中的结点个数
    - 但求其顶点的入度则需要遍历全部的邻接表
    - 图的邻接表表示并不唯一
- **十字链表**
  - 十字链表是有向图的一种链式存储结构。在十字链表中,对应于有向图中的每条弧有一个结点,对应于每个顶点也有一个结点
  - 时间复杂度O(|V| + |E|)
  - 弧结点域结构
    - 尾域（tailvex）弧尾
    - 头域（headvex）：弧头
    - 链域（hlink）：指向弧头相同的下一条弧
    - 链域（tlink）：指向弧尾相同的下一条弧
    - info 域：指向该弧的相关信息

弧结点

tailvex	headvex	hlink	tlink	info

  - 顶点结点域结构
    - data域存放顶点相关的数据信息
    - firstin和firstout 两个域分别指向以该顶点为弧头或弧尾的第一个弧结点

顶点结点

data	firstin	firstout

  - 只能存有向图
- **邻接多重表**
  - 邻接多重表是无向图的另一种链式存储结构
  - 时间复杂度O(|V| + |E|)
  - 边结构
    - mark为标志域,可用以标记该条边是否被搜索过
    - ivex和jvex为该边依附的两个顶点在图中的位置
    - ilink指向下一条依附于顶点ivex的边;
    - jlink指向下一条依附于顶点jyvex 的边
    - info为指向和边相关的各种信息的指针域

mark	ivex	ilink	jvex	jlink	info

  - 顶点结构
    - data域存储该顶点的相关信息
    - firstedge域指示第一条依附于该顶点的边

data	firstedge

  - 只能存无向图

6.3图的遍历

- **广度优先搜索**
  - 广度优先搜索(BFS)
    - 类似于二叉树的层序遍历算法
    - 算法思想
      - 首先访问起始顶点v,接着由v出发,依次访问v的各个未访问过的邻接顶点w1,w2...,wi
      - 然后依次访问w1,w2,,wi的所有未被访问过的邻接顶点
      - 再从这些访问过的顶点出发,访问它们所有未被访问过的邻接顶点,直至图中所有顶点都被访问过为止
    - 利用队列实现搜索
  - BFS算法的性能分析
    - 在最坏的情况下,空间复杂度为O(|V|)
    - 时间复杂度
      - 邻接表存储方式： 顶点：O(|V|) 边：O(|E|) 总时间复杂度：O(|V|+|E|)
      - 邻接矩阵存储方式: 总时间复杂度:
  - BFS算法可以解决最短路径问题
  - 广度优先生成树
    - 在广度遍历的过程中，我们可以得到一棵遍历树，称为广度优先生成树
    - 唯一性
      - 邻接矩阵存储表示是唯一的，故其广度优先生成树也是唯一的
      - 由于邻接表存储表示不唯一，故其广度优先生成树也是不唯一的
- **深度优先搜索**
  - 深度优先搜索（DFS）
    - 类似于树的先序遍历。搜索策略是尽可能"深"地搜索一个图
    - 算法思想
      - 首先访问图中某一起始顶点v,然后由v出发,访问与v邻接且未被访问的任一顶点w1,再访问与w1邻接且未被访问的任一顶点 w2...重复上述过程。
      - 当不能再继续向下访问时,依次退回到最近被访问的顶点,若它还有邻接顶点未被访问过,则从该点开始继续上述搜索过程,直至图中所有顶点均被访问过为止
    - 递归算法 需要借助一个递归工作栈
  - DFS算法的性能分析
    - 空间复杂度为O(|V|)
    - 时间复杂度
      - 邻接矩阵 时间复杂度
      - 邻接链表
        - 找所有顶点的邻接点所需的时间为O(|E|)
        - 访问顶点所需的时间为O(|V|)
      - ***总时间复杂度：o（|v|+|e|）***
  - 深度优先的生成树和生成森林
    - 对连通图调用DFS才能产生深度优先生成树
    - 非连通图产生的是深度优先生成森林
    - ***基于邻接表存储的深度优先生成树是不唯一的***
- **注意**
  - 基于邻接矩阵的遍历所得到的DFS序列和BFS序列是唯一的
  - 基于邻接表的遍历所得到的DFS序列和BFS序列是不唯一的
- **图的遍历与图的连通性**
  - 图的遍历算法可以用来判断图的连通性
  - 对于无向图
    - 若无向图是连通的，则从任一结点出发，仅需一次遍历就能够访问图中的所 有顶点
    - 若无向图是非连通的，则从某一个顶点出发，一次遍历只能访问到该顶点所在连通分量的所有顶点，而对于图中其他连通分量的顶点，则无法通过这次遍历访问
  - 对于有向图
    - 若无向图是连通的，从初始点到图中的每个顶点都有路径，则能够访问到图中的所有顶点
    - 若无向图是非连通的，不能访问到所有顶点

6.4图的应用(上)

- **最小生成树**
  - 一个连通图的生成树包含图的所有顶点，并且只含尽可能少的边
    - 若砍去它的一条边，则会使生成树变成非连通图
    - 若给它增加一条边，则会形成图中的一条回路
  - 最小生成树：权值之和最小的那棵生成树, 则称为最小生成树
  - 最小生成树性质
    - 最小生成树不是唯一的，即最小生成树的树形不唯一
    - 其对应的边的权值之和总是唯一的，而且是最小的
    - 最小生成树的边数为顶点数减1
  - 最小生成树算法
    - Prim算法
      - 概述
        - 始时从图中任取一顶点加入树T,此时树中只含有一个顶点
        - 之后选择一个与当前T中顶点集合距离最近的顶点,并将该顶点和相应的边加入T,每次操作后T中的顶点数和边数都增 1
        - 以此类推,直至图中所有的顶点都并入T,得到的T就是最小生成树。此时T中必然有n-1条边
      - 时间复杂度 $O(|V|^2)$
      - 适用于求解边稠密的图的最小生成树
    - Kruskal 算法
      - 概述
        - 初始时为只有n个顶点而无边的非连通图T={V,{}},每个顶点自成一个连通分量
        - 然后按照边的权值由小到大的顺序,不断选取当前未被选取过且权值最小的边,
        - 若该边依附的顶点落在T中不同的连通分量上,则将此边加入T
        - 否则舍弃此边而选择下一条权值最小的边
        - 以此类推,直至T中所有顶点都在一个连通分量上
      - 时间复杂度 $O(\log|E|)$
      - 适合于边稀疏而顶点较多的图
- **最短路径**
  - Dijkstra算法求单源最短路径问题
    - 辅助数组
      - dist[]:记录从源点v0到其他各顶点当前的最短路径长度,它的初态为:若从v0到vi有弧,则dist[i]为弧上的权值;否则置dist[i]为无穷大
      - path[]:path[i]表示从源点到顶点i之间的最短路径的前驱结点
    - 实现过程
      - 1）初始化:集合S初始为{0},dist[]的初始值dist[i]=arcs[0][i],i=1,2,...,n-1
      - 2)从顶点集合V-S中选出vj,满足 $\mathrm{dist}[j]=\mathrm{Min}\{\mathrm{dist}[i] \mid v_i \in V-S\}$ vj就是当前求得的一条从v0出发的最短路径的终点,令 $S=S\cup\{j\}$
      - 3）修改从V0出发到集合V-S上任一顶点vk可达的最短路径长度:若dist[j]+arcs[j][k]<dist[k],则更新dist[k]=dist[j]+arcs[j][k]
      - 4）重复2）~3）操作共n-1次,直到所有的顶点都包含在S中
    - 时间复杂度 $O(|V|^2)$
    - 不适用于边权值存在负数的情况
  - Floyd算法求各顶点之间最短路径问题
    - 实现过程
      - 初始时,对于任意两个顶点vi和vj,若它们之间存在边,则以此边上的权值作为它们之间的最短路径长度
      - 若它们之间不存在有向边,则以无穷大作为它们之间的最短路径长度
      - 以后逐步尝试在原路径中加入顶点k（k=0,1,2....n-1）作为中间顶点
      - 若增加中间顶点后,得到的路径比原来的路径长度减少了,则以此新路径代替原路径
    - 时间复杂度 $O(|V|^3)$
    - 允许图中有带负权值的边,但不允许有包含带负权值的边组成的回路
    - 适用于带权无向图

6.4图的应用(下)

- **有向无环图描述表达式**
  - 有向无环图：若一个有向图中不存在环，则称为有向无环图，简称DAG图
  - 有向无环图是描述含有公共子式的表达式的有效工具
- **拓扑排序**
  - AOV网概述
    - 若用DAG图表示一个工程,其顶点表示活动,用有向边<Vi,Vj>表示活动Vi必须先于活动进行的这样一种关系,则将这种有向图称为顶点表示活动的网络,记为AOV网
    - 活动Vi是活动Vj的直接前驱,活动Vj是活动Vi的直接后继
    - 这种前驱和后继关系具有传递性,且任何活动不能以它自己作为自己的前驱或后继
  - 拓扑排序定义
    - 每个顶点出现且只出现一次
    - 若顶点A在序列中排在顶点B的前面,则在图中不存在从顶点B到顶点A的路径
  - 拓扑排序实现方法
    - ①从 AOV 网中选择一个没有前驱的顶点并输出
    - ②从网中删除该顶点和所有以它为起点的有向边
    - ③重复①和②直到当前的AOV网为空或当前网中不存在无前驱的顶点为止.后一种情况说明有向图中必然存在环
  - 注意
    - 入度为零的顶点,即没有前驱活动的或前驱活动都已经完成的顶点,工程可以从这个顶点所代表的活动开始或继续
    - 若一个顶点有多个直接后继,则拓扑排序的结果通常不唯一
    - 若各个顶点已经排在一个线性有序的序列中,每个顶点有唯一的前驱后继关系,则拓扑排序的结果是唯一的
    - 生成AOV网的新的邻接存储矩阵,可以是三角矩阵
    - 对于一般的图来说,若其邻接矩阵是三角矩阵,则存在拓扑序列;反之则不一定成立
    - 拓扑排序、逆拓扑排序序列可能不唯一
    - 若图中有环，则不存在拓扑排序序列/逆拓扑排序序列
- **逆拓扑排序**
  - 具体实现
    - DFS算法
  - 思想
    - ①从AOV网中选择一个没有后继（出度为0）的顶点并输出
    - ② 从网中删除该顶点和所有以它为终点的有向边
    - ③重复①和②直到当前的AOV网为空
- **关键路径**
  - AOE网概述
    - 在带权有向图中，以顶点表示事件，以有向边表示活动，以边上的权值表示完成该活动的开 销（如完成活动所需的时间），称之为用边表示活动的网络
  - AOE与AOV
    - 相同点
      - AOE网和AOV网都是有向无环图
    - 不同点
      - AOE网中的边有权值；而 AOV网中的边无权值
  - AOE网性质
    - 只有在某顶点所代表的事件发生后，从该顶点出发的各有向边所代表的活动才能开始
    - 只有在进入某顶点的各有向边所代表的活动都已结束时，该顶点所代表的事件才能发生
  - 关键路径与关键活动
    - 关键路径：从源点到汇点的所有路径中,具有最大路径长度的路径
    - 关键活动：关键路径上的活动
  - 变量含义
    - 事件vk的最早发生事件ve(k)
      - ve(源点)=0
      - ve(k)=Max{ve(j)+Weight(vj,vk)) },vk为vj的任意后继,Weight (vj,vk)表示<vj,vk>上的权值
    - 事件vk的最迟发生事件vl(k)
      - vl(汇点)= ve（汇点)
      - vl(k)=Min{vl(j)-Weight(vk,vj)},vk为vj的任意前驱
    - 活动ai的最早开始时间 e(i)
      - 该活动弧的起点所表示的事件的最早发生时间。若边<vk,vj>表示活动ai,则有e（i）=ve(k)
    - 活动ai的最迟开始时间l(i)
      - 该活动弧的终点所表示事件的最迟发生时间与该活动所需时间之差。若边<vk,vj>表示活动ai,则有l(i)=vl(j)-Weight(vk, vj)
    - 一个活动ai的最迟开始时间l(i)和其最早开始时间e(i)的差额d(i)=l（i)-e(i)
  - 关键路径的算法步骤
    - 从源点出发,令ve(源点)=0,按拓扑有序求其余顶点的最早发生时间 ve()
    - 从汇点出发,令vl(汇点）=ve（汇点）,按逆拓扑有序求其余顶点的最迟发生时间vl()
    - 根据各顶点的ve()值求所有弧的最早开始时间e( )
    - 根据各顶点的 vl()值求所有弧的最迟开始时间l()
    - 求AOE 网中所有活动的差额d(),找出所有d()=0的活动构成关键路径
  - 注意
    - 关键路径上的所有活动都是关键活动,是决定整个工程的关键因素,因此可通过加快关键活动来缩短整个工程的工期
    - 不能任意缩短关键活动,因为一旦缩短到一定的程度,该关键活动就可能会变成非关键活动
    - 网中的关键路径并不唯一,只有加快那些包括在所有关键路径上的关键活动才能达到缩短工期的目的
    - 若关键活动耗时增加，则整个工程的工期将增长
    - 缩短关键活动的时间，可以缩短整个工程的工期
    - 当缩短到一定程度时，关键活动可能会变成非关键活动